A Mathematical Analysis of an Example Delay Tolerant Network using the Theory of Sheaves

Alan Hylton
NASA Glenn Research Center

Robert Short
NASA Faculty Fellow, Glenn Research Center

Robert Green and Metin Toksoz-Exley
American University

***Abstract*— NASA's High-Data Rate Architecture (HiDRA) project is working towards a general yet practical toolkit and knowledge base to help usher in the era of new technologies for space systems communications, such as optical links. The High-Rate Delay Tolerant Networking (HDTN) implementation falls under the umbrellas of both the toolkit and the knowledge base, as its advancements illuminate more general areas of Delay Tolerant Networking (DTN) that need growth. The goal of this paper is to explore the usage of particular mathematical machineries, namely temporal flow networks and sheaves, to identify fundamental, underlying structures in DTN for space systems.**

Satellites, space assets, ground stations, etc. give rise to a disconnected network, and it is the goal of DTN to glue disparate links together into a cohesive system, that is, a network. Depending on a given link, the latencies might be beyond that which the Transmission Control Protocol (TCP) can handle, and contact times might have one-way light times in excess of minute (sometimes significantly longer). Some links might be periodic (say, due to orbital mechanics) or they might not be. This diversity has made it difficult to probe the underlying structure. An immediate consequence is that DTNs in practice today are controlled by globally distributed contact plans (schedules), which are the input to the contact graph routing (CGR) algorithm. While this is effective for smaller networks, it will be very difficult to scale for future networks. Deeper and more rigorous theory is needed to bring DTN to the next evolutionary step.

To this end, this paper introduces and suggests a mathematical framework for DTN, and applies it to a space network that is simulated using an orbital analysis toolkit. The tag-line for the structure known as sheaves is that they are the mathematically precise way of gluing local data together into unique, global data. If we consider routing, we see that networking is a "sheafy" science. We then discuss a simplified sheaf model, known as the cellular sheaf. The sheaf-theoretic analysis is presented and discussed, as it is hoped that this and related papers will help form the primordial ooze of DTN theory. Finally there is a section of future work suggesting follow-on research.

1. INTRODUCTION

In several senses, the *need* for networking in space grows like entropy. Here, we consider two such ways: assets are beginning to enjoy optical (laser) links in addition to the traditional radio frequency (RF) links, and there are more assets in space than ever before. The current optical communication technology demonstrations promise to deliver an order-of-magnitude increase in bandwidth [1] [2] [3], which implies both the choice of different types of links, and the need for the current space and ground infrastructure to adapt concomitantly. Having RF and optical communications implies different kinds of links, meaning even between two assets there could be multiple paths, and the number of assets implies end-to-end paths might feature multiple hops. Given the disruptions and disconnections caused by, for example, mobility, and the wide variance of latencies due to the physical distances between any two given assets, an alternative to terrestrial networking was created to turn the collection of disparate, time-varying links into a *network*. This is the goal of Delay Tolerant Networking (DTN) [4] and its implementations [5]. DTN research is still on-going, and as certain barriers are overcome, others gain prominence. Classically, performance was one such hurdle: it is the goal of this paper to begin work towards how we might achieve greater scalability and routing.

We do not attempt to provide a deep introduction to DTN in this paper - for suitable material, the reader is encouraged to consult [4] [5] [6] [7] [8] - however we do recall the necessities in this paragraph. Architecturally, delay-tolerance is achieved through the use of an overlay network. This overlay is a logical construct that can be thought of as a graph that exists on top of a number of existing assets and links. A vertex in this graph is a place at which data may exit the overlay: such a destination may map to either one physical asset or a collection of many (e.g. an entire constellation). An edge in this graph represents a logical link between two endpoints: such edges may be constructed upon any protocol, known as convergence layers. These consist of small transport-layer protocol adaptations which allow the atom of DTN data, the bundle, to pass through individual edges; note that a bundle may be of arbitrary size. The protocol by which such bundles are transported through the overlay itself is called the Bundle Protocol [5]. A restatement of the goal of this paper is that we bring more rigor and mathematical structure to bear on DTN in order to begin teasing out the fundamental theory of DTN; it is within and by this theory that the authors believe scalability will be realized.

In order to live up to its purpose, achieving scalability of communications, networking must not incur inefficiencies so as to become a bottleneck. Initial work with DTN showed promise in the domain of disconnected networks, however the limitations are well-published [9] [10] [7] [8]; indeed, given representational network traffic, data rates taper off in the low 100's of megabits per second regardless of the power of the underlying hardware. Consider a simple bundle transfer between two nodes: if the bundle is large and sent over TCP/IP, the data rate should be the line-rate. However, we expect most bundles to either be smaller (on the order of the maximum transmission unit (MTU) size), or to be fragmented to such a size. Therefore, we must consider the bundle overhead, and indeed the computational complexity is non-trivial for such reasons as the header fields are not of fixed width, precluding random access. This and implementation specifics have resulted in the mentioned limitations. The need to overcome these bottlenecks gave rise to several performance-optimized DTN implementations, such as High-

Rate DTN (HDTN) and Bundle Protocol library (bplib) [11]. We hasten to add that different implementations, such as the reference implementation, Interplanetary Overlay Network (ION) [12], are designed for particular environments and use-cases; they may all have a place in a single given network given their unique attributes.

HDTN has been shown to process on the order of 10's to 100's of thousands of bundles per second on hardware suitable for Low Earth Orbit (LEO) spacecraft, and to support links over 1 gigabit per second with bundles less than 1 kilobyte in size [7]. We note that it is critical to have several independently developed manifestations of a protocol for interoperability testing. In particular, having two or more helps find bugs in the specification as well as any given code base. Interoperability, however, means more than specification conformancy, or "speaking the same language," and includes not speaking more quickly than one can listen. Hence work was conducted to verify interoperability between ION, HDTN, and bplib over a simulated satellite scenario using a variety of line rates [8].

While the work is not done yet, HDTN and bplib show great promise towards their performance goals, so here we turn our attention to other matters. In a typical DTN, the nodes use InterPlanetary Network (IPN) naming: a name is of the form `ipn:x.y`, where the number `x` is the name of the node and the number `y` refers to the service number [13]. The only topology on a collection of names is the trivial (indiscrete) topology - as such, no hierarchy or ordering is possible; this can be contrasted with Internet Protocol (IP) addressing. The bundle routes are often computed using schedules, known as *contact graphs* [6]. A contact graph contains such information as link start and stop times, bit rates, and one-way light times. These schedules are globally distributed across the network, and a modified Dijkstra's algorithm is used to determine the end-to-end paths. The lack of topological separation and schedule creation, maintenance, and distribution render DTN nearly inapplicable to modern systems. Indeed, pending as of the date of publication, SpaceX has submitted paperwork to the International Telecommunication Union (ITU) for up to an additional 30,000 satellites on top of the 12,000 it already has approval for [14]. These satellites are in LEO, in a decaying orbit, and naturally will be launched over time. Moreover, a system of this magnitude will encounter loss. These factors force any network to be adaptive, bucking against the rigidity exhibited by DTN. The issues are particularly exacerbated by the tendency to have "humans in the loop," that is, manually created schedules.

The primary representation of a network is a *graph* - a collection of edges and vertices (nodes and links) - upon which we apply probability, statistics, game theory, and more [15]. We also note that a TCP connection presupposes such conditions as end-to-end connectivity, which cannot be assumed in a DTN. Consider a TCP/IP connection: given low latencies and an end-to-end paths, intermediary buffering requirements are relatively low. On the other hand, one way of considering DTN is as a store-carry-and-forward network: given a period of disconnection, bundles (of arbitrary) size must be buffered. Hence as we add information to our graph, we have to consider buffering capabilities. Other approaches to DTN have incorporated this information into graph representations of the network in a variety of ways [16] [17] [18] [19] [20] [21]. While each approach functions well as a model for DTN in the specific applications, there is a lack of a unified theoretical foundation for DTN as a graph.

One goal of this paper is to present the structure that forms a unified theory of DTN. As this theory takes form, more deterministic approaches towards algorithm development, implementation, management, and so forth become possible. More specifically, if topological structure that persists across typical DTNs becomes known, then a respective addressing scheme may be developed. Indeed, improved scalability will be a natural by-product of taking the leap from naming to addressing. Below, we introduce the structures germane to DTNs.

2. TEMPORAL FLOW NETWORKS

This section is meant to set up the mathematical tools in use for our graph theoretic model of DTNs. In order to be able to properly model the complexities of DTN, even given the confines of space networks, we must develop a rigorous definition of the mathematical objects that represent them. This is a crucial step to be able to bring existing tools from higher mathematics to bear on the problems at hand. With that in mind we define a temporal flow network (TFN), which are introduced in [22]. We recall that a graph $G = (V, E)$ consists of a set of vertices V and a collection E of (possibly directed) edges between them.

We note that the below definitions are technical, but there are graphical examples - see Figures 1 & 2.

Definition 2.1 (Temporal Graph)**.** We call a graph $G = (V, E)$ a temporal graph if:

- G is a directed graph; and
- for each edge $e \in E$, we assign a finite set $L_e \subset \mathbb{N}$ of labels identifying the integer time instances when e is available.

This is denoted $G = (V, E, L)$ where $L = \bigsqcup_{e \in E} L_e$ to summarize the necessary information.

Once we have established a temporal graph, if we want to use it to model flow we must add additional structure to make it a flow network. First, we recall that in a contact graph, link durations and average bit rates are known, and hence link capacities are known. Individual nodes will be equipped with known storage (buffering) capabilities.

Definition 2.2 (Temporal Flow Network)**.** A temporal flow network G is a temporal graph equipped with:

- an identified source vertex s and sink (target) vertex t;
- a capacity function $c : E \to \mathbb{R}$; and
- a buffer function $B : V \to \mathbb{R}$, where $B(s)$ and $B(t)$ are assumed to be infinite.

This is denoted $G = (V, E, L, s, t, c, B)$ to summarize the necessary information.

One can think of a temporal flow network as a sequence of graphs at each time instance connected by edges mapping each vertex to its copy in the next time instance with buffer

capacities representing the storage capacities over the edges between time instances.

Definition 2.3 (Time-Extended Graph)**.** Let $G = (V, E, L, s, t, c,$ be a temporal flow network. Then its corresponding time-extended graph $G^{TE} = (V^{TE}, E^{TE})$ is a weighted directed graph where:

- for each vertex $v \in V$, there are $\ell^{TE} + 1$ copies in V^{TE} denoted $v_0, v_1, \ldots, v_{\ell^{TE}}$ where ℓ^{TE} is the largest integer in $\bigcup_{e \in E} L_e$
- for each edge $e = (v, w) \in E$ and each $i \in L_e$, there is an edge $e^{TE}_{vw,i} = (v_{i-1}, w_i) \in E^{TE}$
- for each vertex $v \in V$, there is an edge $e^{TE}_{v,i} = (v_{i-1}, v_i) \in E^{TE}$
- there is a capacity function $c^{TE} : E^{TE} \to \mathbb{R}$ defined by:

$$c^{TE}(e^{TE}_{x,i}) = \begin{cases} c(e) & \text{if } x \in E \\ B(v) & \text{if } x \in V \end{cases}$$

To simplify our computations, we can simplify the time-extended graph for a given temporal flow network by collapsing edges that only serve to extend the buffer for the vertex. To state this precisely, if the only edges containing v_i in G^{TE} are $e^{TE}_{v,i}$ and $e^{TE}_{v,i+1}$, then we can remove v_i, $e^{TE}_{v,i}$, and $e^{TE}_{v,i+1}$, and replace them with (v_{i-1}, v_{i+1}) with capacity $c^{TE}((v_{i-1}, v_{i+1})) = B(v)$. We call these simplified time-extended graphs (STEGs) and as they are computationally simpler and theoretically justifiable, they will be used instead of time-extended graphs for the remainder of this paper.

The main approach we implement in this paper involves using these STEG objects as an abstraction of a potential space network and then to model the maximum information flow over these graphs. This, while in its current form is an unrealistic representation of the situation, is a proof of concept that easily allows for future more detailed approaches to be developed.

Toy Examples

In this subsection we will work through a few manageable examples.

The first example is purely artificial. Let G be the graph shown in Figure 1. Let S and T be the source and sink nodes. Let the gray numbers inside each intermediary node be the buffer capacity, the black number by the edges be the edge capacity, and the sets next to each edge be the set of times the edge is available for flow.

If we wish to calculate the maximum temporal flow, we cannot use traditional max flow algorithms. However, we can form a STEG of this graph which is shown below in Figure 2, and then apply existing max flow approaches. This example is simple enough to calculate the max flow by hand even with a non systematic approach.

The rules of max flow are discussed later in this section and we leave the computation of max flow as an exercise to the reader (see Figure 6).

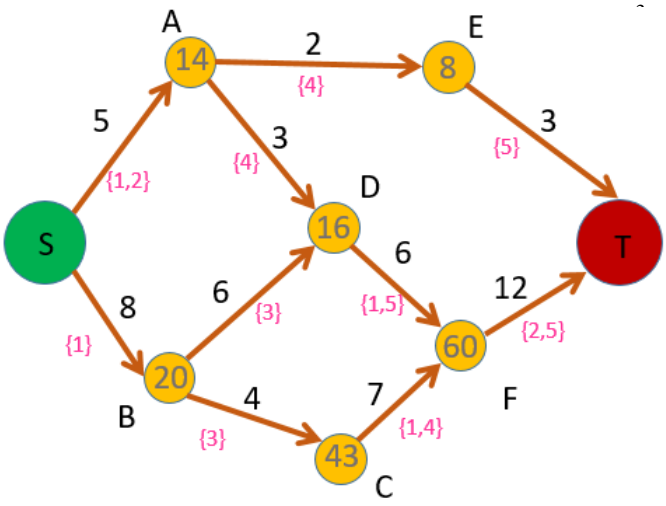

Figure 1. An example temporal flow network

For this paper, code was written to provide a framework within which experimentation with TFNs can be conducted. For the purposes of computing max flow over the TEG, it is possible to skip creating the TFN and go straight to creating the STEG: this is the approach taken by the authors. The code will be considered further in subsection 2. First, we illustrate this using an example social network where the constituents are the authors. We assume that communications depends on peoples' whereabouts over time, and that we might have no three (or four) in the same room at the same time. We can model this as a temporal graph. Depending, perhaps, on how quickly one speaks, different "data rates" are exhibited. If we further assume that one person must deliver information to another, we create a TFN. The schedule that might be used as input to the program is shown in Figures 3 & 4.

The information shown in Figures 3 & 4 is used to directly generate the STEG shown in Figure 5.

Data Generation and Programming Approach

We used the graph-tool [23] package in Python which is built to perform many algorithms and computations within the envelope of graph theory, using high efficiency C++ implementations using a Python wrapper. This was used to both compute the max flow over the graph and to create visualizations.

In order to put our STEG code through its paces, we generated relatively large systems. The means was orbital analysis software. This includes, e.g., Satellite Orbital Analysis Software and Systems Tool Kit (STK). NASA is developing the Strategic Center for Networking, Integration, and Communications (SCENIC) tool, which also simulates the orbital patterns and communication capacities of various NASA assets [24]. By creating scenarios using such tools, we can induce a measure of realisticness as well as direct applicability. This will be explored further in Section 3.

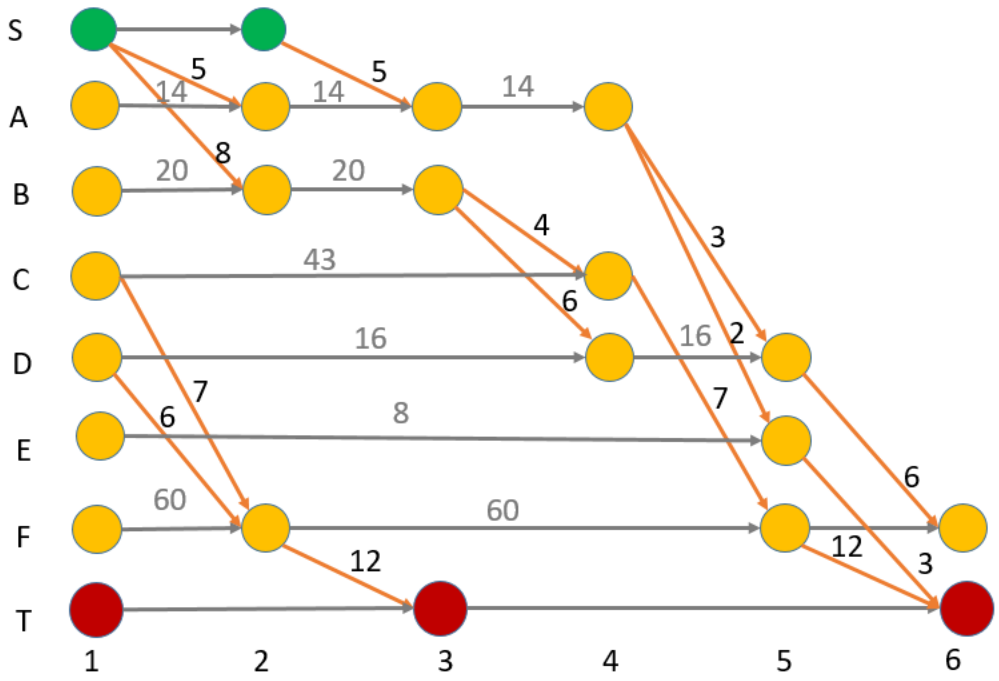

Figure 2. The STEG associated to the temporal flow network shown in Figure 1

	Bob	Met	Rob	Alan
Bob	0	1.25	0.75	4
Met	1.25	0	1.1	0.85
Rob	0.45	1.71	0	0.05
Alan	1.9	2.2	2.8	0

Figure 3. Data rate input to the program (social network example)

Bob		Met		Rob		Alan		Rob		Bob	
start		end		start		end		start		end	
0.2		0.6		0.05		0.3		0		0.4	
0.8		1.7		0.8		1.8		0.5		1.9	
2.5		2.9		2		2.6		4		5	
3		3.6		3		3.2					
4		4.4		3.6		4.1					

Figure 4. Contact schedule input to the program (social network example)

The first and main piece of pertinent data from SCENIC and STK are times line-of-sight availability between different space assets. We used this along with a time slice window, which was used to create a STEG based on the line of sight data. The assumption is if two assets have line of sight at the beginning of the window then it will persist throughout the window and data can be transferred over that whole window. other inputs that the program needs in order to calculate the max flow is the amount of data that can be stored at each asset and an estimated capacity for each channel. We then developed a proof of concept Python package to take the STK and SCENIC data and build our data structures which then can be used as input to the graph-tool package. The authors plan to determine that possibility of open-sourcing the code.

Max Flow

As discussed, one mathematical approach for analyzing networks as a whole that has proven historically very useful and productive is graph theory. It can be used to do cost benefit analysis of different utilization of shipping networks, or to find the most efficient path from one's computer to Google. One problem that has been heavily studied in this field is that of max flow, the aim of which is to find the maximum amount of units that can flow across a given network. Other examples of the broad applicability include determining the flow of water through a system of pipes and the flow information through a network of computers. The question of max flow over a disconnected network, say from space assets to Earth, boils down to solving max flow problem over a graph theoretic representation of the network in our new setting. First we should understand max flow on a traditional graph theoretic representation of a network before we bring it to the Temporal flow networks we use to analyze DTN.

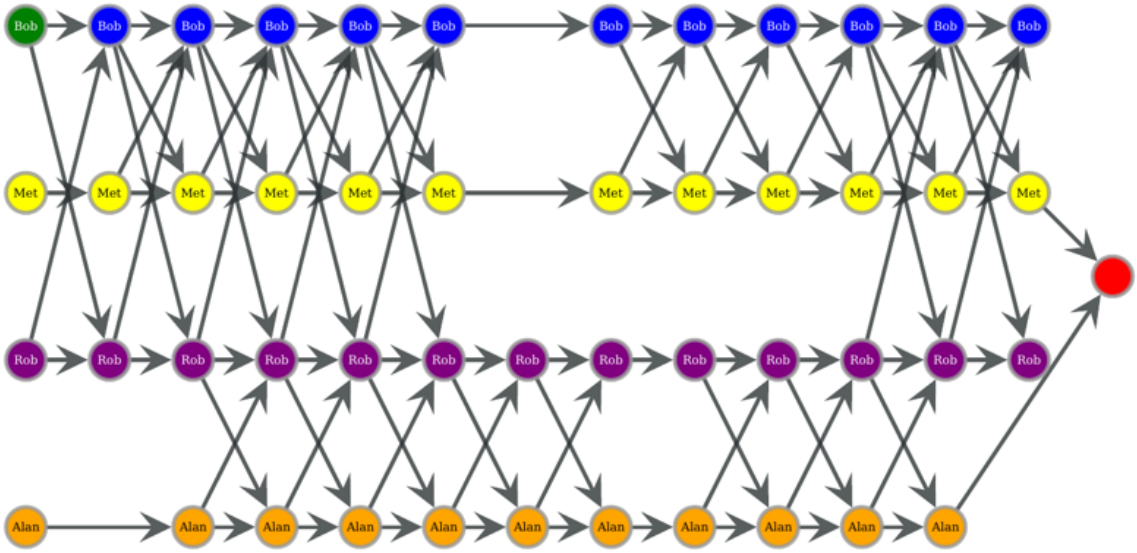

Figure 5. STEG associated to social network example. Here we are considering the maximum flow from bob to metin and alan from time 0 to five with time step of .4

When solving max flow, one is presented with a graph $G = \{V, E, s, t, c\}$, where V is the set of vertices, which represent nodes in the graph. E is the set of edges which connects those vertices and can also be thought of as a proper subset of $V \times V$. The source s is a special vertex where the data units to be transferred through the graph originate. The sink, or target, is t, and is another special vertex, serving as the destination of the units traveling through the graph. Lastly c is a function from E to $\mathbb{R}^+$ denoted by $C(u, v)$ which indicates the capacity of the directed edge (u, v). With this graph construction we can define a flow as follows.

A flow is a mapping $f : E \to \mathbb{R}^+$ denoted by $f(u, v)$ subject to the following two constraints:

- $f(u, v) \leq C(u, v) \; \forall \; (u, v) \in E$; and
- $\sum_{u:(u,v)\in E} f(u, v) = \sum_{u:(v,u)\in E} f(v, u)$ for each $u \in V \setminus \{s, t\}$.

The first rule amounts to a formal capacity constraint. The second rule eliminates the possibility for any flow to originate or terminate anywhere except for the source and the sink. With this in mind we can now define the value of an $s-t$ flow or a flow from s to t by $|f| = \sum_{v:(s,v)\in E} f(s, v)$. Now that we can determine the value of a flow, the natural progression is to maximize it.

Computing max flow is a problem that has been of great interest to graph theorists for a long time, and with such interest comes many different approaches to computing it. All of them involve systematically testing out different flow options, but the approach varies from algorithm to algorithm. This variance in approaches can change what the computational complexity will depend on. However as graphs get very large computational complexity quickly becomes a non-trivial concern. As this paper is meant as a proof-of-concept the authors have decided to forgo lengthy discussion of optimization. The algorithms used by our Python package to maximize flow over a given network are standard ones of the field that where already built into the graph-tool package referenced above.

3. Applications to Space Networks

Both STK and SCENIC were utilized to construct proof-of-concept models that we could run our analysis on. We started by determining what format the data from STK and SCENIC and constructed our package around building a model that would accept these formats as inputs. For both SCENIC and STK the data we were getting was in terms of line-of-sight data for our models between specific assets in the model. For SCENIC, the data came in the form of universal coordinated time (UTCG) which we had to convert in our code to a standard format for consistency with STK.

The first model we set up was a system designed to simulate the Artemis program, which is scheduled to build a long-term lunar capability with several relay satellites and the Lunar Reconnaissance Orbiter (LRO) to communicate back to Earth; see Figure 7. For this model, we constructed four assets in SCENIC, including a moon station but leaving out the Earth communications. We designed the system so that the lunar base would communicate with the two ARTEMIS 1 and ARTEMIS 2 relay satellites that, between their two orbits, had a near-constant line-of-sight with the lunar base. Then between the two ARTEMIS satellites, communications can be established with LRO when possible. The data we collected from SCENIC was over a period of a year and two months. This was used as input into our code and the STEG that emerged was indicative of the lack of direct connectivity between the lunar base and LRO while also providing a useful

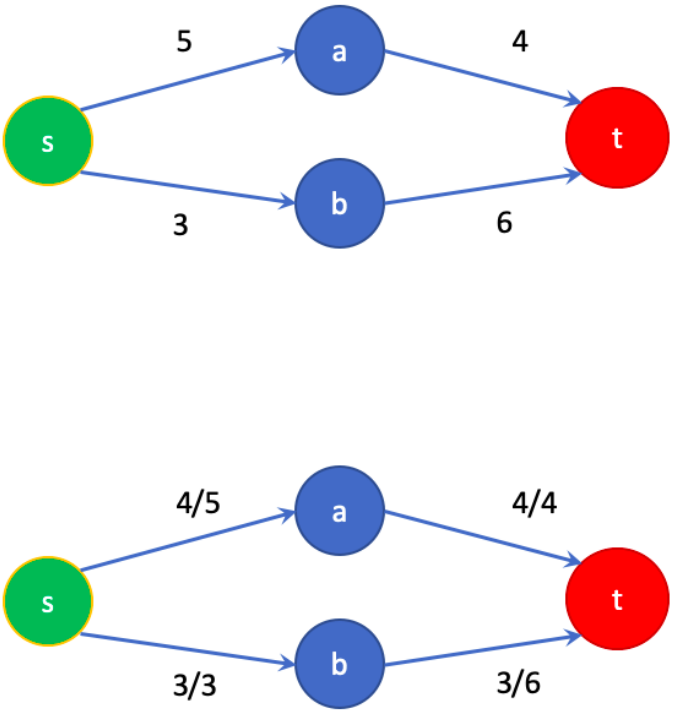

Figure 6. Max flow example: The top is the capacities, and the bottom is the max flow

visualization of the frequency and time of communication that the model achieved.

Just as it was with the model, creative license was taken with the RF parameters: the values chosen are not particularly relevant to determining the tractability of the algorithms. As such, our max-flow code gave us an estimate of 168.3 TB being maximally communicated over 61 days. This amount, while large in a theoretical model, demonstrates the value of thinking of the network capabilities of a system of information. The corresponding STEG is shown in Figure 8.

The second model we set up was a larger model that included a lunar relay, a lunar rover, a Mars relay, a Mars rover, the ISS, and three different ground stations, Goldstone CA, Canberra, and Madrid, as illustrated in Figure 9. This system worked similarly to the smaller Artemis model wherein the assets on the ground communicated through the relays around their moon or planet via a relay satellite network in geosynchronous orbit (namely, Tracking and Data Relay Satellite System (TDRSS)). This larger model simulated something closer to existing NASA networks except instead of singular points of communication with each asset we were attempting to construct a larger model. Some of the assets in the model such as the ISS, LRO, the Mars relay, and the ground stations were all accurate in terms of location and orbits. We note the Mars relay was really Mars Atmosphere and Volatile EvolutioN (MAVEN). While there are existing rovers on Mars and the moon, for this model, we used fictional rovers so that we could match hardware requirements to the communications assets we desired. With that in mind, we made some modifications to the hardware used on some relays, satellites, and ground stations including adding antennas/bands simply to ensure compatibility between all of our assets since we wanted to have as many connections as possible. Getting data on this larger model stretched SCENIC to its capacities since it was such a large model with so many orbits and assets. Thus we had to collect data for only a month with much less frequent intervals to check on the connections. Nevertheless, our STEG for this model had 45,794 vertices and 239,783 edges in our graph of connections. We note that on an AMD Threadripper 2990wx that was shared among many users, computations took place on the order of minutes.

Our two models were similar in inspiration but different in terms of scale and complexity. The smaller lunar-centric model was designed to be a more realistic model in terms of mimicking what the Artemis program is hoping to accomplish, while the larger model was much more focused on extending our network as wide as possible with as many different types and locations of assets as possible. Both of these models demonstrated that we can visualize and calculate maximum flow on the networks which are useful analytic tools to understand, optimize, and improve networks, both existing and planned. The networks were theoretical in that we had to create new hardware capabilities and modify existing hardware, however the modifications made were not egregious. To make sure we were running our code on as complex a model as we could create in our time-frame, we ensured there was enough connectivity in the model to have multiple channels for information to flow from every asset and that every asset could theoretically communicate with any other asset (though not necessarily directly). Being able to run our models through visualization and calculate max-flow proved that we can glean actionable insights about networks from this analysis.

4. SHEAVES

The idea of transmitting data over a network relies upon us having some notions of the type of data we are studying. Whether this is traditional bit strings or higher-level quantitative or qualitative data, we need an effective way of describing this data. Moreover, data traveling over a network needs to remain consistent, no matter the sender or receiver of the information.

As an example, the data we need to define a temporal flow network needs to be consistent over the course of the interval we are considering. While our model works well if connections can be established and speeds are constant, a more sophisticated model might require individual connection inputs and variable speeds depending on location, time, and other factors. In addition, if we want to construct more sophisticated networking structures for DTN, a framework for both describing this data and comparing this data would be essential.

In mathematics, one method for collating local data into a consistent piece of larger global data comes in the form of a *sheaf*. We will define a sheaf below in terms of categorical language. The value to doing this is that no matter what category our data belongs to, we will still be able to use a sheaf to construct the appropriate data structure and model. Sheaves have been studied extensively in mathematics; in recent years they have been reconstructed explicitly to handle discrete data structures as would be found in networks. More general introductions to sheaves in applied topology can be

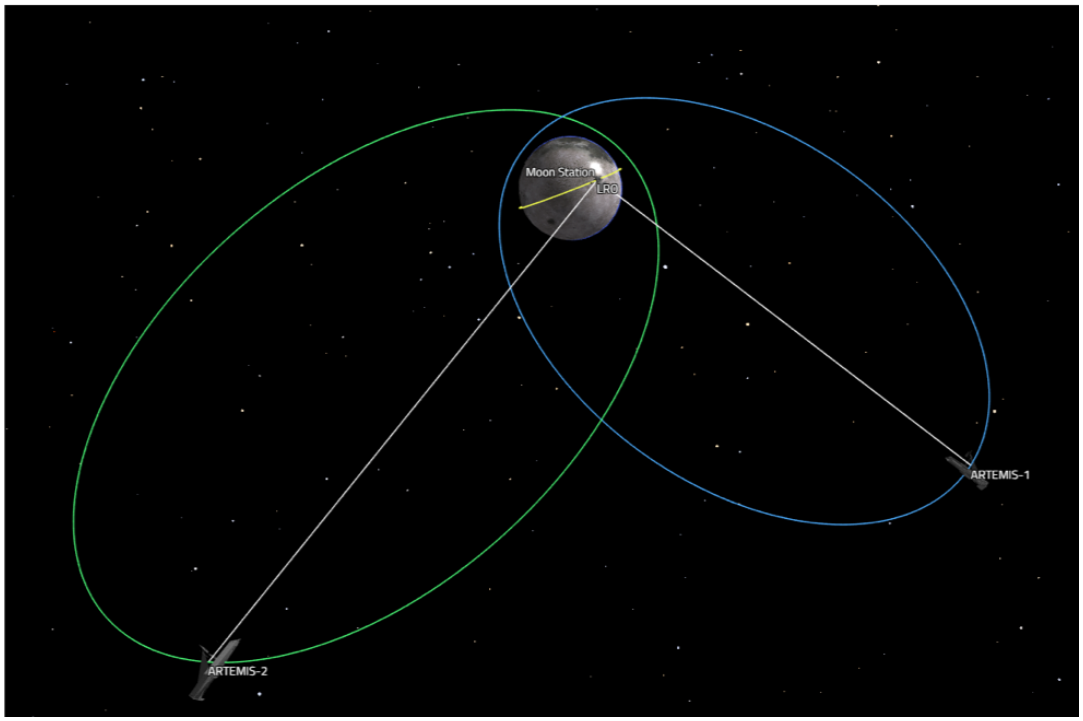

Figure 7. Simplified Artemis-lunar model

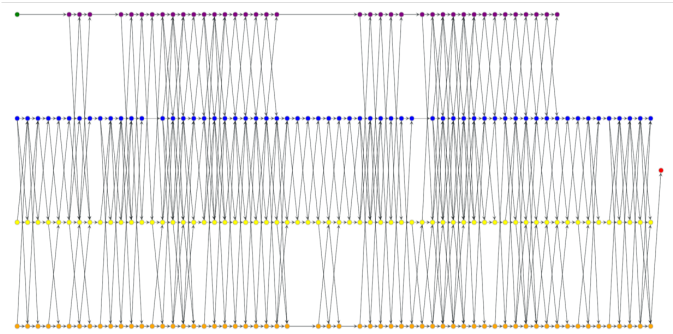

Figure 8. Visualization of our lunar-centric model. Blue represents Artemis 1, yellow represents Artemis 2, purple represents the Lunar station, orange represents the LRO, green is the source (the Lunar station), and red is the sink (the LRO communicating to Earth)

found in [25] while technical details can be found in [26] [27] [28] [29] [30].

Sampling theory has been generalized and described using morphisms of sheaves; this theory counts Shannon-Nyquist theory as a special case [31]. In sampling, one essentially attempts to reconstruct a function from individual samples; the function is the global data, and the samples are the local data. We see analogous processes in networking as well, particularly in routing. Distance vector routes use local connectivity information - tables based on neighbors. Link state routers send their tables about their neighbors to all routers, and hence can make routing decisions based on global data. This provides some intuition on why networks are "sheafy," also see [32].

One reason to consider sheaves in DTN models is that sheaves can organize information and relate it across a full network. As an example, the information we impose to the base graph in a temporal flow network can be thought of as a sheaf. The buffer capacities and edge capacities we associate to the network are not strictly part of the network graph structure, but they form an essential piece for analysis of DTN. The value of including this information as a sheaf is that sheaves structurally include the relationships between the different pieces of information. However, sheaves are more general than this: they can incorporate quantitative or qualitative data that could be relevant to networking problems being considered.

For our purposes, we will use a cellular sheaf, which is a type of sheaf designed for use over graphs, of which our STEGs are examples. In the subsequent definition, think of the graph G as a network (or a STEG), and think of the category $\mathcal{C}$ as a mathematical descriptor for the type of data being applied to the network.

Definition 4.1. A *cellular sheaf* F on a graph $G = (V, E)$ to a category $\mathcal{C}$ consists of assignments of

1. objects $F(v) \in \mathrm{ob}(C)$ to each vertex $v \in V$ (each called the *stalk* over v);

2. objects $F(e) \in \mathrm{ob}(C)$ to each edge $e \in E$ (each also called the *stalk* over e);

3. arrows $F(v, e) \in \mathrm{Hom}(F(v), F(e))$ for $v \in V$ and $e \in E$

Figure 9. Our larger model

whenever v is adjacent to e (each is called the *restriction map* from e to v).

The sheaf can be thought of as the abstract space in which the information we want lives. If we want a specific instantiation of that data, we select a consistent representation of our data within the sheaf. This selection, when consistent across the entire network, is called a global section of the sheaf.

As mentioned above, the data needed to form a temporal flow network can be thought of as a sheaf. Given a network G, which is merely the vertices and edges connecting the vertices, a temporal flow network requires capacity values over each edge, buffer values over each vertex, and a set of active time intervals over each edge. We can define this as a sheaf by $F(v) = \mathbb{R}$ and $F(e) = \mathbb{R} \times \mathcal{P}(\mathcal{T})$, where T is the set of time intervals and $\mathcal{P}(\mathcal{T})$ is the collection of subsets of time intervals. The restriction maps here allow us to provide additional information or relationships between our values. We could impose the restriction that our buffer values must exceed incoming capacity values, or that sufficiently low capacity values are not associated to especially small time intervals.

If we make our restriction maps so that the buffer values must be greater than the sum of incoming capacity values, a section of this sheaf is represented in Figure 1. Notice that individually, the values are valid elements of our sets, but the key to the section is that the restriction maps are held consistent across the entire network. This consistency of the data with respect to the rules of the network is the key desirable property of sheaves.

As these global sections are desirable quantities, we frequently want to analyze or compare different global sections. The analysis of global sections has traditionally fallen to sheaf cohomology, denoted $H^*(-; F)$ for a sheaf F. Within a cohomological framework, it is possible to compare or even combine different global sections to identify different equivalence classes. Depending on the sheaf and structure in the data category $\mathcal{C}$, sheaf cohomology can describe several different features of the global sections.

As an example of this, Robert Ghrist and Yasuaki Hiraoka introduced a sheaf to model network codings over a network [27]. They used sheaf cohomology as a means of computing maximum flow when data is moving through a network in the presence of a network coding.

Network codings are useful in manipulating how data moves through a network in an attempt to improve efficiency. The key feature of a network coding is a coding map telling how to encode information at one vertex as it is transmitted to another vertex. To be able to productively use this information in a sheaf, Ghrist and Hiraoka define network codings in the following way:

Definition 4.2. A network coding on a directed graph $G = (V, E)$ is a directed graph along with:

- an identified source vertex s and sink (target) vertex t;
- a capacity function $c : E \to \mathbb{R}$; and
- for each edge $e = (v, w)$, a local coding map $\phi_e : \mathbb{R}^{\kappa(\tilde{\approx})} \to \mathbb{R}^{()}$ where $n(s)$ is the amount of information coming out of the source, and $n(v) = \sum_{e \in \mathrm{In}(v)} c(e)$ for $v \neq s$.

Note that $\mathrm{In}(v)$ is the set of edges which point at v, and we denote a graph equipped with a network coding by $G = (V, E, s, t, c, \{\phi_e\})$.

To encode a network coding as a sheaf, the structure of the network coding gives us a reasonable category of data, namely vector spaces with linear maps between them. Thanks to how the local coding is defined, there are natural choices for the data types over each vertex and each edge. In addition, the local coding maps form natural restriction maps for some of the vertex-edge pairings. To define a network coding sheaf, we need only specify data types of each vertex and edge and codify the receiving coding maps as projection maps in

the appropriate dimensions. Doing this yields the following definition for a network coding sheaf:

Definition 4.3. A network coding sheaf F on a directed graph $G = (V, E, s, t, c, \{\phi_e\})$ equipped with a network coding assigns

- to each vertex $v \in V$, $F(v) = \mathbb{R}^{\kappa(\tilde{\approx})}$;
- to each edge $e \in E$, $F(e) = \mathbb{R}^{()}$;
- a linear map $F(v, e) : F(v) \to F(e)$ whenever v is adjacent to e such that:

$$F(v,e) = \begin{cases} \text{proj} : F(v) \to F(e) & \text{if } e \in \text{In}(v) \\ \phi_e : F(v) \to F(e) & \text{if } e = (v, w) \text{ for some } w \in V \end{cases}$$

What Ghrist and Hiraoka discovered was that the maximum flow of information through a network coding was intimately connected to the sheaf cohomology structure in the network coding sheaf. In particular, they prove the following theorem:

Theorem 4.4. *[27] For a network coding sheaf F on network X, $H^0(X; F)$ is equivalent to the set of information flows on the network.*

From here, we can determine that the maximum flow of information over a network coding sheaf F is the dimension of $H^0(X; F)$. This is a reasonably computable quantity, and can permit us to compare network codings from a maximum flow perspective.

The computability aspect is a key point to reflect upon. One big advantage to using sheaves to model data is that there is software designed to handle sheaf-theoretic computations. Namely PySheaf [33], a software package developed by Michael Robinson et al. from American University. PySheaf uses linear algebra and abstract algebra to construct and compare sheaves and sheaf-like objects. This software makes computing sheaf information tractable, even for large or complex constructions. In the future, we hope to find ways to model sheaves over a DTN by incorporating PySheaf computations into our TFN construction.

5. Conclusion and Future Work

Temporal Flow Networks form an excellent graph-theoretic model for Delay-Tolerant Networking. There is sufficient structure in TFNs for the computation of the maximum amount of information that can flow over the network. Moreover, our model can be constructed in a way that predicts connectivity, even for large and complicated networks, over long periods of time. In addition to TFN's ability to model DTNs, powerful mathematical machinery can be placed on top of them. Uncovering better tools for DTN research gave rise to several projects that we hope to pursue.

Temporal Flow Networks

TFNs themselves need to be generalized so that they can be used to build traditional DTN algorithms. We expect that there is a means of transcribing existing routing algorithms, including contact graph routing [6], into the language of TFNs. We also expect that our definitions here rely upon assumptions that may not hold in all DTN cases. A sufficient generalization of TFNs should be able to represent any theoretical or practical Delay-Tolerant Network without needing to impose additional restrictions.

Data-rates, and thus link capacities, should vary with time. While CGR assumes that average rates per contact are specified in order to determine queuing depths, some links (depending on latency) might support reactive mitigation to, e.g., loss. In a given contact, instantaneous data rates can change, for example, with proximity.

TFNs can also be extended to better respect latency issues. TCP, for example, is highly sensitive to latency and packet loss [34]. There is value in rigorously determining the appropriateness of a given protocol given link conditions.

In a network, we should consider multiple simultaneous source and receiver nodes. Therefore, in a natural setting, it might be impossible to truly determine buffer requirements. This might be mitigated in practice by the use of network policies. However, to perform computations using our TFN models, buffer capacities were assumed to be large enough to be a non-factor.

Incorporating cost-benefit optimization into max flow can help make the beginnings of a graph-theoretic analysis of different protocols.

Structure and Protocols

The idea of identifying and relating methods in DTN via sheaf theory is very appealing. The idea of modeling information consistently flowing over a network as a sheaf is not new, but the specific implementations that would be relevant and generalizable to DTN still need to be developed. Pieces of this have been developed for different aspects of networking. For example, Sanjeevi Krishnan in [30] found a sheaf that could be used to compute maximum flow over any network, without referencing or requiring a network coding structure.

The intrinsic structure of a DTN will likely have a much finer granularity than the extrinsic structure, that is, a given DTN will naturally take on more roles than as prescribed by naming schemes. However, this structure is unknown. Topological data analysis (TDA) has enabled sub-classification of previously prescribed types in data sets, see [35] to see an example regarding sub-types of type 2 diabetes. It might be that sheaves could be used to discuss not only the local robustness of networks, but also the roles that nodes play in a network.

If some structure of a network is determined, we might be able to simplify max flow calculations for the temporal network by compacting certain sub networks, for example, by having a Mars network represented by one node from an Earth-node's perspective. Such simplifications would be beneficial from a computational perspective, but it could also lead to understanding relationships between networking structures at different granularities. Sheaves, again, would provide good theoretical structures for this comparison. Another mathematical tool to consider is discrete Morse theory (DMT) [36] [37]. DMT might be useful for directing flows, but is

particularly useful for dimension-reduction (simplification). If portions of a network could be collapsed by DMT, it would illuminate the local hierarchies of those sub-networks.

Some protocols, such as TCP and the Datagram Congestion Control Protocol (DCCP) have built-in congestion control techniques. TFNs might be used to bridge not only max-flox but also congestion control into DTNs.

Finally, Robinson in [28] describes a sheaf that models protocols effectively and provides a means of resolving conflicts or interference in sheaf-based networking. These form pieces of a much larger project among mathematicians, namely developing multiple related sheaves that could be used to model all of the layers to networking. One direction to pursue over a long period of time would be to adapt sheaves that could model all layers in the Open Systems Interconnection (OSI) model for traditional networking. A sheaf-theoretic OSI model may be reasonable to transport to the DTN setting. We are certainly interested in pursuing this, but there is significantly more work that needs to be done before this can be effectively approached.

Software

Another project is pure software development. We hope that future researchers are able to improve upon our algorithms so that the community is even more comfortable using TFNs as a framework. We also want to work on stronger methods for importing data into our software so that data from a wide range of modeling software.

Other potential contributions include simply deriving better algorithms, or writing better implementations. Also, one might begin transcribing DTN algorithms into the language of TFNs. Presumably, a more user-friendly library combining PySheaf and graph-tool, or building upon existing software in a more accessible program, would entice more researchers to join.

Miscellaneous

In the course of sheaf discussions, we noticed they might also apply to space navigation. Celestial navigation often utilizes computationally expensive star trackers, which compare pictures of a visible star field to a known catalog of stars. This method is very effective, but if navigational information needs to be updated between star tracking algorithm refreshes, sheaves might enable sensor fusion between either accelerometers and camera data or accelerometers and star-tracker output. This might be particularly true given sheaf-sampling of signals with a high signal-to-noise ratio and available power spectral density profiles of given space craft.

6. ACKNOWLEDGEMENT

We wish to acknowledge NASA SCaN for their continuing support, in particular Dr. Donald Cornwell and Badri Younes. Dr. Michael Robinson's guidance with sheaves and PySheaf was indispensable. We also thank Timothy Gallagher and Dr. Daniel Reliable for their contributions to our work and to the NASA internship program and faculty fellowship program that made this work possible.

REFERENCES

[1] D. V. Murphy, J. E. Kansky, M. E. Grein, R. T. Schulein, M. M. Willis, and R. E. Lafon, "LLCD operations using the Lunar Lasercom Ground Terminal," in *Free-Space Laser Communication and Atmospheric Propagation XXVI*, H. Hemmati and D. M. Boroson, Eds., vol. 8971, International Society for Optics and Photonics. SPIE, 2014, pp. 250 – 256. [Online]. Available: https://doi.org/10.1117/12.2045509

[2] D. J. Israel, B. L. Edwards, and J. W. Staren, "Laser communications relay demonstration (lcrd) update and the path towards optical relay operations," in *2017 IEEE Aerospace Conference*, March 2017, pp. 1–6.

[3] A. Biswas, M. Srinivasan, R. Rogalin, S. Piazzolla, J. Y.-C. Liu, B. C. Schratz, A. Wong, E. Alerstam, M. W. Wright, W. T. Roberts, J. M. Kovalik, G. Ortiz, A. Na-Nakornpanom, M. D. Shaw, C. Okino, K. S. Andrews, M. Y. Peng, D. S. Orozco, and W. M. Klipstein, "Status of nasa's deep space optical communication technology demonstration," *2017 IEEE International Conference on Space Optical Systems and Applications (ICSOS)*, pp. 23–27, 2017.

[4] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "RFC 4838, Delay-Tolerant Networking Architecture," *IETF Network Working Group*, 2007. [Online]. Available: https://tools.ietf.org/html/rfc4838

[5] K. Scott and S. Burleigh, "RFC 5050, Bundle Protocol Specification," *IETF Network Working Group*, 2007. [Online]. Available: https://tools.ietf.org/html/rfc5050

[6] G. Araniti, N. Bezirgiannidis, E. Birrane, I. Bisio, S. Burleigh, C. Caini, M. Feldmann, M. Marchese, J. Segui, and K. Suzuki, "Contact graph routing in dtn space networks: overview, enhancements and performance," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 38–46, March 2015.

[7] A. Hylton, D. Raible, and G. Clark, "A delay tolerant networking-based approach to a high data rate architecture for spacecraft," in *2019 IEEE Aerospace Conference*, March 2019, pp. 1–10.

[8] A. Hylton, D. Raible, G. Clark, R. Dudukovich, B. Tomko, and L. Burke, "Rising above the cloud – toward high-rate delay-tolerant networking in low-earth orbit," in *37th AIAA International Communications Satellite Systems Conference*, October 2019, pp. 1–8.

[9] P. Muri and J. McNair, "A performance comparison of dtn protocols for high delay optical channels," 04 2013, pp. 183–188.

[10] S. Schildt, J. Morgenroth, W.-B. Pöttner, and L. Wolf, "Ibr-dtn: A lightweight, modular and highly portable bundle protocol implementation," *Electronic Communications of the EASST*, vol. 37, 2011. [Online]. Available: https://journal.ub.tu-berlin.de/eceasst/article/view/512/544

[11] J.-P. Swinski and et alii, "bplib," https://github.com/nasa/bplib, 2018.

[12] S. Burleigh, "Interplanetary overlay network: An implementation of the dtn bundle protocol," *2007 4th IEEE Consumer Communications and Networking Conference*, pp. 222–226, Jan 2007.

[13] L. Clare, S. Burleigh, and K. Scott, "Endpoint naming for space delay / disruption tolerant networking," in

2010 IEEE Aerospace Conference, March 2010, pp. 1–10.

[14] M. Wall, "Spacex's starlink constellation could swell by 30,000 more satellites," Oct 2019. [Online]. Available: https://www.space.com/spacex-30000-more-starlink-satellites.html

[15] M. E. J. Newman, *Networks an introduction*. Oxford University Press, 2018.

[16] M. H. Jain and R. Patra, "Implementing delay tolerant networking," *University of California, Berkeley, Computer Science Division, Berkeley, CA*, vol. 94720, 2003.

[17] S. Jain, K. Fall, and R. Patra, *Routing in a delay tolerant network*. ACM, 2004, vol. 34, no. 4.

[18] E. Magistretti, J. Kong, U. Lee, M. Gerla, P. Bellavista, and A. Corradi, "A mobile delay-tolerant approach to long-term energy-efficient underwater sensor networking," in *2007 IEEE Wireless Communications and Networking Conference*. IEEE, 2007, pp. 2866–2871.

[19] E. P. Jones and P. A. Ward, "Routing strategies for delay-tolerant networks," *Submitted to ACM Computer Communication Review (CCR)*, 2006.

[20] H. Ntareme, M. Zennaro, and B. Pehrson, "Delay tolerant network on smartphones: Applications for communication challenged areas," in *Proceedings of the 3rd Extreme Conference on Communication: The Amazon Expedition*. ACM, 2011, p. 14.

[21] L. Wood, W. D. Ivancic, W. M. Eddy, D. Stewart, J. Northam, C. Jackson, and A. daSilvaCuriel, "Use of the delay-tolerant networking bundle protocol from space," 2009.

[22] E. C. Akrida, J. Czyzowicz, L. Gasieniec, L. Kuszner, and P. G. Spirakis, "Flows in temporal networks," *CoRR*, vol. abs/1606.01091, 2016. [Online]. Available: http://arxiv.org/abs/1606.01091

[23] T. P. Peixoto, "The graph-tool python library," *figshare*, 2014. [Online]. Available: http://figshare.com/articles/graph{_}tool/1164194

[24] NASA, https://www1.grc.nasa.gov/space/scan/scas/scenic/, 2019.

[25] R. Ghrist, *Elementary Applied Topology*. CreateSpace Independent Publishing Platform, 2014.

[26] J. Curry, "Sheaves, cosheaves and applications," 2013.

[27] R. Ghrist and Y. Hiraoka, "Applications of sheaf cohomology and exact sequences to network coding," *Proc. NOLTA*, 2011.

[28] M. Robinson, "Modeling wireless network routing using sheaves," 2016.

[29] ——, "Hunting for foxes with sheaves," *Notices of the American Mathematical Society*, vol. 66, pp. 661–676, 5 2019.

[30] S. Krishnan, "Flow-cut dualities for sheaves on graphs," 2014.

[31] M. Robinson, "A sheaf-theoretic perspective on sampling," 2014.

[32] R. Ghrist, "Data aggregation over networks via integration," The 5th IEEE International Conference on Autonomic Computing, jun 2008, keynote.

[33] M. Robinson, C. Capraro, and B. Praggastis. (2016) The pysheaf library. [Online]. Available: https://github.com/kb1dds/pysheaf

[34] "Tcp network latency and throughput," Spirent, Tech. Rep., 2016.

[35] L. Li, W.-Y. Cheng, B. S. Glicksberg, O. Gottesman, R. Tamler, R. Chen, E. P. Bottinger, and J. T. Dudley, "Identification of type 2 diabetes subgroups through topological analysis of patient similarity," *Science Translational Medicine*, vol. 7, no. 311, pp. 311ra174–311ra174, 2015. [Online]. Available: https://stm.sciencemag.org/content/7/311/311ra174

[36] R. Forman, "A user's guide to discrete morse theory," *Sém. Lothar. Combin.*, vol. 48, 12 2001.

[37] N. A. Scoville, *Discrete Morse Theory*. American Mathematical Society, 2019.